Basic Effects:

- Scene Graph
    - Objects
        - All animated objects in the scene are manually composed models
        - The ship is composed of two parts
        - The fish's tail is animated separate from its body
        - The top line of the hook elongates while the bottom remains the same
    - Theory
        - The objects in a scene are organized hierarchically. The world is the base node, and the objects in it are child and grandchild nodes. Only the base node's render function is called; the child nodes are rendered when the graph is traversed.
    - Implementation
        - We used the SGNode class given in framework.js
        - The using the .append() function, we attached the skybox, water, terrain, fish, and boat to the root node. The hook was then attached to the boat.
    - Issues
        - Though certain objects have portions that are separately animated, that animated was not accomplished using the scene graph. Given more time, we could have added objects to the boat and animated those objects as well.
- Materials
    - Objects
        - The fish, hook, boat, water, and terrain are all rendered as materials that use textures
    - Theory
        - Any object can be rendered as a material with ambient, specular, diffuse, emission, and shininess properties. These properties effect how it appears when illuminated. When the object is also textured, these properties and blended with the texture.
    - Implementation
        - We used the MaterialSGNode class given in the framework
        - With this, we passed in various texture nodes and a render node as the children of the MaterialSGNode.
        - We then defined certain properties of the material, such as ambience
    - Issues
        - We do not have any objects that are colored using only material properties. All of our objects use textures along with their material properties. Given more time, other objects could be added to the boat that are colored purely through material properties.
- Texturing
    - Objects
        - All objects in the scene other than the light objects use textures

            - A skybox
            - The boat
            - The fish
            - The hook
            - The terrain
            - The water
    - Theory
        - A point on a given texture can be mapped to a point on a given object. That point on the object will then have the color of the respective point on the texture. In this way, the full object will appear as though it has been painted with the colors of the texture.
    - Implementation
        - We used the AdvancedTextureSGNode provided in the framework, which allowed us to pass in the texture, texture unit number, texture uniform, and any child nodes.
            - Based on the texture uniform and other specially defined variables, the textures of the water and terrain could be changed and blended.
        - The skybox used the EnvironmentSGNode class that we defined ourselves. The initialization function for the textures was also written by us.
- Illumination
    - Objects
        - One light source can be seen near the top of the scene. This acts as a ‘sun’
        - Another light source is attached to the boat and only illuminates objects within a certain angle. It behaves like a spotlight
    - Theory
        - A light can be attached the scene and it generates its own ambient, specular, and diffuse properties that work with any material properties. Illumination is created based on the position of the light, the surface normal, and the position of the camera. The shader programs assign color to a surface based on the illumination and material properties blended with any vertex colors or texture colors.
    - Implementation
        - The LightSGNode class provided in the framework allowed us to generate our two light sources. One acts as the ‘sun’ and was attached to a sphere object in the scene graph and placed near the top of the scene. The ‘sun’ illuminates all objects in the scene; the function for this is in the terrain fragment shader. The second light sources acts as a ‘spotlight’. It was also attached to a sphere object which was attached to the front of the boat. This ‘spotlight’ only illuminates objects within a specific cone-shaped area. This is accomplished in the terrain fragment shader; a normal phong

shader function was adjusted to only over a certain area. Since this light is attached to the boat, it also moves with the boat.

- Transparency
    - Objects
        - The water surface is transparent.
    - Theory
        - An alpha value is used to determine how transparent the object is. The alpha value is essentially blended with the color of the fragment to create transparency.
    - Implementation
        - This is implemented in the terrain fragment shader. When fragColor is finally given a value (after computing the texture color and the lighting properties), it is the value of the color of the fragment (as an rgb value computed from the lighting properties) and an alpha value.
    - Issues
        - Given more time, we would also place a transparent plane in front of the camera whenever the camera goes underwater. The plane would have been blue, had an animated surface, and always faced the camera and moved with the camera. This would be to simulate being underwater.
- Camera
    - Movement
        - Upon loading the page, automatic camera movement directs the user through the different scenes
        - If ‘c’ is pressed, the camera is initialized in a default location and the user can use their mouse and keyboard to move the camera and trigger the object animations
    - Theory
        - A given camera position, lookAt vector, and Up vector will determine the ModelView matrix, which determines what is projected onto the computer screen. When the camera is moved or when it is directed to look in a different direction, the a different perspective of the scene should be seen (as determined by the ModelView matrix)
    - Implementation
        - Global variables cameraPosition, cameraUp, and cameraGaze are used to compute the View matrix. This computation is done in two functions called calculateViewMatrix() and calculateCameraPostion().
            - CalculateViewMatrix() calculates the perspective of the scene based on the current elapsed time and generates automatic camera flight
            - CalculateCameraPosition() calculates the perspective of the scene based on keyboard and mouse interaction. This is used during the manual camera mode.

        - Essentially, the cameraUp, cameraGaze, and cameraPostion (or up, eye, center) and multiplied together to produce a mat4 that is used as the View matrix.
    - Issues
        - When the user enters manual camera mode during the 30 second animation, the time elapsed for the animation does not stop. So, if they switch back to automatic camera later, the animation is over.
        - When manual camera mode is entered, the user only gets one chance to trigger the individual object animations.

Special Effects:

- S1. Terrain from Heightmap
    - Appearance
        - The terrain can be seen throughout the scene; it looks like hills.
    - Theory
        - A heightmap can be used as a texture on a plane which is then used in the vertex shader to determine where the vertices of the plane are. The heightmap is just a grayscale image where the black areas represent low points and the white areas represent high points. Each point on the heightmap corresponds to a point on the plane and the color of the heightmap corresponds to the height of a point on the plane. Once these points are computed, the fragments are then drawn to create a solid surface.
    - Implementation
        - This is implemented in the terrain vector shader in a function called renderTerrain(). Using a grayscale heightmap, the height of a point on a plane is computed using the color of the point on the map. An offset is then applied to the computed vector and this final value is used as a vertex on the plane. The normal vectors between the vertices are then computed to be used later.
    - Issues
        - The area of the terrain is a bit small, so it's very easy to accidentally travel outside of the world.
- S6. Animated Water Surface
    - Appearance
        - The water surface can be seen throughout the scene. The surface has waves, is slightly transparent, and reflects the environment
    - Theory
        - A heightmap can be used as a texture on a plane which is then used in the vertex shader to determine where the vertices of the plane are. The heightmap is just a grayscale image where the black areas represent low points and the white areas represent high points. Each point on the heightmap corresponds to a point on the plane and the color of the heightmap corresponds to the height of a point on the plane. Once these

points are computed, the fragments are then drawn to create a solid surface. Elapsed time is then used to alter the texture (which then affects the height of the plane vertices) so as to give the appearance of waves.

- Implementation
  - This is implemented in the terrain vector shader in a function called renderWater() and main. A grayscale heightmap is altered based on elapsed time. The varying height of a point on the plane is computed using the varying color of the point on the map. An offset is then applied to the computed vector and this final value is used as a vertex on the plane. The varying height of a given vertex creates the wave motion.
- Issues
  - Even though there are waves on the surface of the water, the lighting under the water is not effected. There is no refraction or wobbling of light underwater in our scene. Given a more sophisticated shading function, this could potentially be fixed.